Ministère de l'Education nationale et de la Formation professionnelle

EXAMEN DE FIN D'ETUDES SECONDAIRES TECHNIQUES

Régime technique – Division technique générale – Section informatique

Session 2012

BRANCHE : PROGRAMMATION

DATE : 24 mai 2012 *DUREE :* 4 heures

Indications pour la réalisation pratique :

- Renommez le dossier **LTAM_13GI_xyz** en **LTAM_13GI_*Numéro*** où *Numéro* est votre numéro d'identification de l'examen (exemple: LTAM_13GI_27).
- Rajoutez votre numéro d'identification ainsi que le numéro du poste dans chaque unité que vous modifiez!

Explications préliminaires :

Lisez l'énoncé attentivement jusqu'à la fin avant de commencer à programmer!

Dans la suite, il s'agit de programmer un petit jeu connu sous le nom 'Snake'. Pour ceux qui ne connaissent pas le jeu, voici une brève description (texte extrait en majeure partie de *Wikipedia.fr*) :

> Le joueur contrôle une longue et fine créature semblable à un serpent, qui doit slalomer entre les bords de l'écran et les obstacles qui parsèment le niveau. Pour gagner chacun des niveaux, le joueur doit faire manger à son serpent un certain nombre de pastilles, allongeant à chaque fois la taille de la bestiole. Alors que le serpent avance inexorablement, le joueur ne peut que lui indiquer une direction à suivre (en haut, en bas, à gauche, à droite) afin d'éviter que la tête du serpent ne touche les murs ou son propre corps. Le niveau de difficulté est contrôlé par l'aspect du niveau (simple ou labyrinthique), le nombre de pastilles à manger, l'allongement du serpent et sa vitesse.

Principe de fonctionnement de la version à réaliser :

Dans notre programme, le serpent peut manger trois types de nourriture :

- Nourriture normale (jaune, +10 points),
- Nourriture mauvaise (rouge, -100 points),
- Nourriture concentrée (bleue, +100 points)

Ouvrez le dossier '**Q1_Snake**'. Vous y trouverez le programme modèle **Modele.exe.** Ouvrez ce programme et faites quelques essais pour vérifier si vous avez bien compris le principe de fonctionnement. Pour changer de direction, utilisez les touches de navigation (flèches) du clavier.

Le programme consiste en trois parties :

- Le programme principal avec la fiche **frmMain** qui contient la fiche sur laquelle le jeu va être dessiné ainsi qu' un composant *Timer*.
- L'unité **UGame** qui contient la classe **TGame** qui effectue la gestion du jeu (serpent, nourriture, mouvements).
- L'unité **ULinks** qui contient la description de la classe **TLink** ("maillon"[1]) et de ses dérivés (**THead**, **TFood**, **TBadFood**, **TPowerFood**) qui sont utilisés pour composer le serpent ainsi que la nourriture.

Le serpent et la nourriture sont réalisés en tant que deux listes chaînées d'objets.

Le terrain de jeu consiste en une grille imaginaire de 30 positions en largeur et 20 positions en hauteur. La première position en haut à gauche porte les coordonnées [0,0], la dernière en bas à droite porte les coordonnées [29,19]. Chaque position peut éventuellement être occupée par un maillon. Dans le programme, les coordonnées X et Y se rapportent toujours à ces positions. Ce n'est que dans les méthodes **Draw** que ces positions sont converties dans des coordonnées en *pixels*.

1 "maillon" en allemand : *Glied einer Kette*

Réalisation :

- **TOUS** les fichiers sont à sauvegarder dans le dossier '**Q1_Snake**'.
- Référez-vous au projet **Modele.exe** pour les détails de la réalisation!
- Dans le dossier '**Q1_Snake**' créez le programme **Snake.dpr** avec l'unité **UMain.pas** dans laquelle se trouve la fiche **frmMain**. Vous ajouterez les unités **ULinks** et **UGame** au même projet.

Unité ULinks [21 points] :

1. Créez une nouvelle unité **ULinks.pas** qui contient d'abord la constante **Size** qui indique la largeur et la hauteur d'un maillon (**TLink**) en pixels. **Size** est initialisé par la valeur 32. Pour permettre des changements ultérieurs, employez **Size** dans la suite du programme et non la valeur numérique 32.

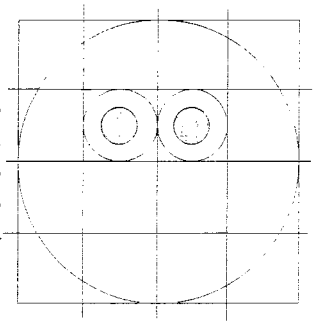
   Définissez dans **ULinks.pas** la classe **TLink** qui est une classe abstraite et qui sert de base à la définition des maillons du serpent et de la nourriture. **TLink** possède les attributs et méthodes suivantes :

Color	attribut privé, couleur du maillon, dépend du type du maillon
Value	attribut privé, valeur (points/score) de ce maillon, dépend du type du maillon
X	attribut public, position horizontale de ce maillon (valeurs possibles : $0 \leq X \leq 29$)
Y	attribut public, position verticale de ce maillon (valeurs possibles : $0 \leq Y \leq 19$)
Next	attribut public, type **TLink**, connexion au prochain maillon dans la liste chaînée
Create	constructeur initialisant les attributs **X, Y** et **Next** par des valeurs reçues comme paramètres. **Create** sera surchargé par les classes dérivées pour initialiser les autres attributs.
Destroy	destructeur qui détruit ce maillon et **récursivement** aussi tous les maillons qui suivent dans la chaîne.
Draw	méthode qui dessine le maillon actuel et **récursivement** aussi tous les maillons suivants dans la chaîne. Le canevas destinataire est fourni comme paramètre. Dans **TLink** sont effectués les actions de dessin communes à tous les types de maillons. Chaque maillon est constitué d'un cercle (diamètre **Size**) de la couleur du maillon qui est dessiné à la position X,Y du maillon. La largeur du stylo est 3 à travers tout le programme. La position à l'écran est calculée en considérant la constante **Size**. La méthode sera surchargée par les les classes dérivées.
getLinkAt	méthode **récursive** qui retourne le maillon (type **TLink**) qui se trouve à la position **pX, pY.** La méthode retourne **nil** si aucun noeud de cette chaîne ne se trouve à cette position. (pX et pY sont les paramètres de la méthode).
getTotalValue	méthode **récursive** qui retourne la valeur totale de tous les maillons dans la chaîne (= somme des attributs **Value**).

Remarque : Si vous avez des difficultés à définir les méthodes de façon récursive, vous pouvez comme ***solution de secours*** définir les méthodes de façon itérative, mais vous n'obtiendrez pas tous les points qui se trouvent sur ces questions. D'autre part les méthodes récursives sont beaucoup plus simples à formuler.

[13 points]

2. Définissez les classes **THead**, **TFood**, **TBadFood** et **TPowerFood** qui sont toutes dérivées de **TLink**. Profitez au mieux de l'héritage et du polymorphisme. Les classes ont les caractéristiques suivantes :



THead : Valeur : 100 points, couleur verte

La tête du serpent porte des 'yeux' qui sont dessinés par deux cercles bleus sur deux cercles blancs. Chaque cercle blanc a un diamètre qui correspond au quart du diamètre du maillon. Le bord inférieur des cercles blancs touche la diagonale centrale du maillon. Les deux cercles se touchent au milieu. Les cercles bleus ont un diamètre qui correspond au huitième du diamètre du maillon et ils sont centrés dans les cercles blancs.

TFood : Valeur : 10 points, couleur jaune, aucun autre signe distinctif

TBadFood : Valeur : -100 points, couleur rouge

Ce type de maillon porte un trait vertical (signe ' – ') au milieu. La couleur du trait est *clMaroon* et sa longueur est la moitié du diamètre du maillon.

TPowerFood : Valeur : 100 points, couleur *clAqua*

Ce type de maillon porte un trait vertical et un trait horizontal (signe ' + ') au milieu. La couleur des traits est *clNavy* et leur longueur est celle de la moitié du diamètre du maillon.

[8 points]

Unité UGame [33 points] :

3. Créez une nouvelle unité **UGame.pas** qui contient d'abord la définition du type **TDirection** :

```
TDirection = (dirUp, dirDown, dirLeft, dirRight);
```

Définissez dans **UGame.pas** la classe **TGame** qui sert à gérer le serpent et la nourriture. Bien que le jeu se situe sur une grille imaginaire de 30x20 cases, il n'existe PAS de structure qui mémorise le contenu de chaque position de la grille. **Les positions des maillons sont uniquement mémorisés dans les maillons même**, ce qui évite une double gestion des positions et facilite la programmation des mouvements. **TGame** possède les attributs et méthodes suivantes :

Snake attribut privé, type **TLink**, contiendra la chaîne de maillons représentant le serpent. Au démarrage le serpent contient uniquement une tête qui se trouve au milieu (position [15,10])

FoodChain attribut privé, type **TLink**, contiendra la chaîne de maillons représentant la nourriture. Au démarrage **FoodChain** contient 14 éléments placés à des positions aléatoires (10 éléments du type **TFood**, 2 du type **TBadFood**, 2 du type **PowerFood**). Veillez à ne pas placer deux éléments à la même position ! Remarque : Au cours du jeu, la liste **FoodChain** garde toujours ces mêmes 14 éléments, ce n'est que leur position dans la grille qui est changée, lorsque l'un d'eux est 'mangé'. (C.-à-d. lorsqu'un maillon est 'mangé', il réapparaît comme nourriture à une autre position à l'écran.)

Direction attribut privé, type **TLink**, contiendra la direction actuelle de la tête du serpent. Au démarrage le serpent se dirigera vers la droite

Collided attribut privé, type **boolean**, valeur ***true*** dès qu'une collision a eu lieu (signale la fin du jeu ; ***false*** au départ – évidemment :)

Create constructeur initialisant les attributs comme décrit ci-dessus

Destroy	destructeur qui supprime le serpent et la chaîne de nourriture
Draw	méthode qui dessine le serpent et la chaîne de nourriture sur le canevas envoyé comme paramètre
setDirection	méthode permettant de changer le contenu de **Direction** (possède un paramètre du type **TDirection**)
hasCollided	retourne la valeur actuelle de **Collided**
getScore	retourne la valeur totale des maillons que possède actuellement le serpent
Move	méthode appelée régulièrement par le programme principal pour effectuer les mouvements du serpent → détails ci-dessous

[10 points]

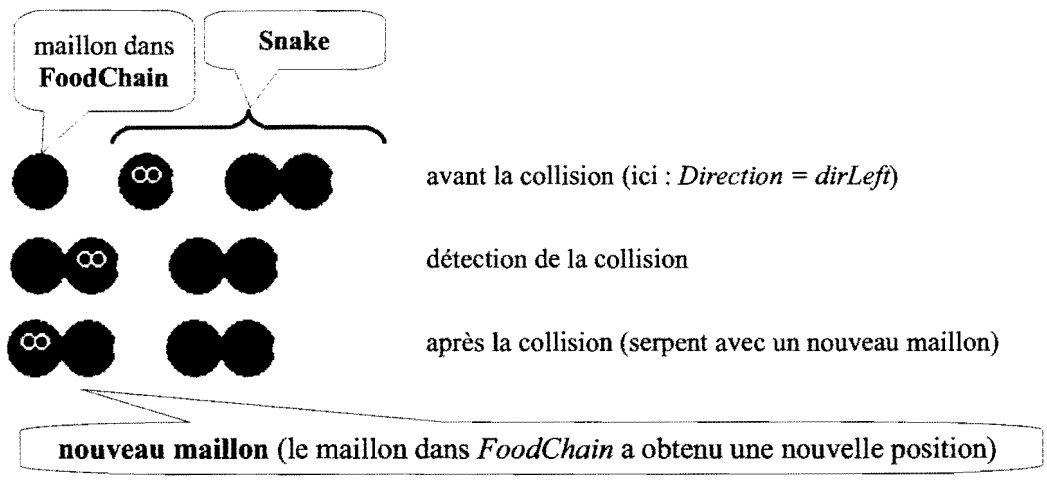
4. Réaliser la méthode **Move** qui fonctionne comme suit :

   La nouvelle position de la tête du serpent est calculée en fonction de **Direction**.

   - Si la nouvelle position se trouve en dehors du terrain de jeu ou sur un maillon du serpent, il y a collision, c.-à-d. tous les éléments gardent leur position actuelle et **Collided** devient ***true***.

   - Si la nouvelle position contient un élément de la chaîne de nourriture, alors la nourriture est 'mangée', c.-à-d. :
     Un nouveau maillon du même type que celui que le serpent a 'mangé' est inséré dans la chaîne, derrière la tête du serpent. Ce nouveau maillon obtient la position actuelle de la tête du serpent. La tête du serpent avance à la nouvelle position. Le reste du serpent reste inchangé. Dans la chaîne **FoodChain**, le maillon en cause obtient une nouvelle position (choisie de façon aléatoire, mais qui n'est pas occupée actuellement).

     Illustration : le serpent 'mange' un maillon de nourriture :



   - Dans tous les autres cas, le serpent avance sans obstacles, c.-à-d. la tête prend la nouvelle position et tous les autres maillons prennent la position du maillon qui précède dans la liste. (Si vous le jugez utile, vous pouvez définir une méthode supplémentaire dans **TGame** ou **TLink** pour réaliser ce transfert des positions).

[23 points]

Unité UMain [6 points] :

5. Complétez le programme principal et sa fiche principale **frmMain** comme suit :

   La fiche porte une couleur de fond verte claire (code couleur $EEFFEE).

   Le programme contient une instance **Game** du type **TGame** qui doit être détruite lorsqu'on ferme le programme.

   A chaque fois que la fiche est redessinée, et après chaque mouvement, le jeu est redessiné et le total des points est affiché dans la barre de titre du programme sous la forme :

   Snake – Score : *<points>*

   Le programme contient un **TTimer** nommé **tmTimer** qui est activé dès le départ et qui répète ses actions 10 fois par seconde. A chaque intervalle de **tmTimer**, un mouvement du jeu est exécuté et il est contrôlé, si le jeu est terminé. Si oui, **tmTimer** est désactivé et un message de fin de jeu est affiché.

   Les changements de direction se font par l'événement **OnKeyDown** sur la fiche en testant sur les valeurs **VK_UP, VK_DOWN, VK_LEFT, VK_RIGHT** du paramètre **Key.** (Si nécessaire, voir à ce sujet l'aide sur '*OnKeyDown*' et '*Virtual Key Codes*').